FACULDADE DE ENGENHARIA DA UNIVERSIDADE DO PORTO

Computação Paralela na Análise de Tráfego de Redes de Comunicação

Tiago Samuel da Rocha Silva

Mestrado Integrado em Engenharia Informática e Computação

Orientador: Prof. Jorge Barbosa

25 de Julho de 2016

Computação Paralela na Análise de Tráfego de Redes de Comunicação

Tiago Samuel da Rocha Silva

Mestrado Integrado em Engenharia Informática e Computação

25 de Julho de 2016

Resumo

As redes informáticas dominam as comunicações e são utilizadas para os mais diversos fins, desde distribuição de conteúdos informativos, vídeo, *e-business*, etc, gerando uma quantidade significativa de pacotes de dados que têm de ser reencaminhados desde o servidor até ao consumidor.

A análise de tráfego de rede é importante para a engenharia de tráfego e para a detecção de intrusões e ataques. Este é apenas um dos muitos exemplos em que a análise de grande volume de dados possui um impacto directo na sociedade e na economia. Um dos problemas mais críticos em manter uma rede de comunicação sob controlo é capturar e analisar o tráfego. A complexidade destas tarefas tem vindo a aumentar ao passo que as redes se têm vindo a tornar iterativamente mais rápidas. Um dos principais problemas reside na capacidade de computação necessária para processar o tráfego de rede capturado, assim como armazenar os dados relevantes dessa mesma análise.

Com o aumento da capacidade dos links e da complexidade das técnicas de análise surge a necessidade de aplicar técnicas paralelas de análise de tráfego, recorrendo a software que permita implementar algoritmos capazes de percorrer todo o fluxo de tráfego e alocar o processamento destes mesmos dados, da forma mais eficiente tendo em conta os recursos disponíveis na altura.

Este trabalho pretende avaliar a viabilidade de implementação dos algoritmos de gestão de tráfego recorrendo à comparação entre *Tensorflow* e outras ferramentas já existentes desenvolvidas para a gestão de tráfego, encontrando um equilíbrio entre flexibilidade de implementação e performance na execução.

Abstract

Computer network dominate communications and are used for the most diverse ends, from digital content distribution, video, e-business, etc, generating a significant amount of data packages that have to be redirected from the server to the consumer.

Network traffic analisys is important for traffic analisys and for attacks and intrusions detection. This is one of many examples that traffic analisys in a great amount of data has a direct impact in society and economy.

One of the most critical problems in maintaining a network communication under control is to capture and analyse traffic. The complexity of these task has been increasing as network have been becoming iteratively faster. One of the main problems lies in the necessary computing capacity to process captured network traffic, as storing relevant data of that same analisys.

With the increase of link capacity and network techniques complexity, the need to apply parallel network traffic analisys techniques is born, resorting to software that allows the possibility to implement algorithms capable of travelling through all traffic flow and allocate the processing of that same data, in the most efficient way considering the available resources at that same time.

This work intends to evaluate the feasibility to implement traffic management algorithms resorting to a new tool called Tensorflow, compared to other existing tools developed to this purpose, findinng the balance between implementation flexibility and execution performance.

Agradecimentos

Gostaria de agradecer aos meus pais, Zeferino e Luzia, pela paciência sem fim, por todos os sacrifícios que sempre fizeram com um sorriso na cara.

Um grande agradecimento aos meus amigos de curso que ao longo destes 5 anos me incentivaram e me ajudaram nas horas mais difíceis.

Um palavra de apreço ao meu orientador, Professor Jorge Barbosa por toda a compreensão e ajuda, ao Professor Ricardo Morla, pela ajuda de orientação e ao Paulo Vaz pela disponibilidade.

E um grande obrigado, por fim a todos os meus amigos que de alguma forma contribuíram

Tiago Samuel Silva

"Do only the things that you're able to justify."

Pedro Alexandre Silva

Conteúdo

CONTEÚDO

Lista de Figuras

Lista de Tabelas

Capítulo 1

Introdução

1.1 Contexto/Enquadramento

Análise de tráfego é o processo de interceptar e examinar mensagens de forma a deduzir informação a partir de padrões na área da comunicação. É crucial para as grandes empresas possuir um bom sistema de análise de tráfego de forma a garantir a integridade e privacidade de toda a sua informação. Ainda assim um bom sistema de analise de tráfego pode ser algo mais complexo do que aquilo que parece. Para além de ser necessário todas as unidades físicas para ser possível analisar localmente aquilo que está a decorrer na nossa rede, é também necessária possuir um Software capaz de lidar com os grandes fluxos de informação que decorrem de um rede de comunicação. Quanto maior o fluxo de informação, mais frequente será a existência de picos de fluxos de informação o que pode levar a grandes atrasos e até perdas de informação [BCA11].

Computadores de alta velocidade e redes telefónicas transportam grandes quantidades de informação e tráfego de sinalização. Os Engenheiros por trás da construção e manutenção destas redes usam uma combinação de *hardware* e ferramentas de *software* para monitorizar o fluxo de tráfego de uma rede de comunicação [MGB15].

Algumas destas ferramentas operam na rede em modo *live*, ao passo que outras armazenam a informação para uma análise *offline* a partir de *software* específico. Uma grande parte das tarefas de análise destas técnicas processa até centenas de *gigabytes* de informação. Isto dificulta grandemente a possibilidade de realização de algumas técnicas em modo offline, pois acaba por se tornar uma quantidade demasiado grande para que compense o seu armazenamento para posterior análise.

Com a consequente necessidade de avanços nas áreas da inteligência artificial, *machine learning*, *data analysis*, vem em paralelo um também consequente avanço na criação de ferramentas de análise por parte das grandes empresas nestas áreas levando a que estejamos constantemente a assistir ao lançamento de novas ferramentas para análise de informação, levando a que muitas vezes um utilizador menos experiente possa ir de encontro a um caso de *information overload*[MCB13].

1.2 Projeto

Este projecto lidou na sua grande maioria com:

- Tensorflow - Software da Google lançada em Novembro de 2015. É apresentado pela Google como uma interface para expressar algoritmos de *machine learning* e uma implementação funcional para executar esses mesmo algoritmos. Mas pode também ser interpretado uma biblioteca *open source* para computação numérica recorrendo a grafos de fluxo de informação. Tendo em conta que é uma ferramente relativamente recente e apesar de bem documentada, ainda não é possível encontrar grande quantidade de informação.[ten16]

- MATLAB - Ferramenta que integra computação, visualização, e programação num ambiente '*user friendly*' em que os problemas e soluções são expressados numa notação relativamente familiar. Desenvolveu alguma relevância a partir de funcionalidades como desenvolvimento de algoritmos, modelação, análise de informação, entre outros.

- R - um ambiente de desenvolvimento integrado para cálculos estatísticos e gráficos, altamente expansível com o uso dos pacotes, que são bibliotecas para funções específicas ou áreas de estudo específicas.

- Python - Linguagem de programação de *high-level*, bastante flexível e de fácil "leitura"que permite o uso de bibliotecas de *Machine Learning/Data Mining/Data Analysis*.

1.3 Motivação e Objetivos

Esta dissertação apresenta como principal objetivo a implementação de diferentes algoritmos *Data Clustering* em diferentes ferramentas plataformas/ferramentas/Linguagens de forma a ser possível obter uma base de comparação à performance do *Tensorflow* na execucação de algoritmos, assim como também poder observar a flexibilidade de programação no *Tensorflow* comparativamente com as outras ferramentas geralmente usadas nesta área.

Sendo a Google uma das principais empresas na área do *Deep Learning* e apesar de o *Tensorflow* ser uma biblioteca *open-source*, não existe praticamente nenhuma informação sobre o seu potencial a nível de performance na execução de algoritmos. Nesse sentido existe a necessidade de realizar testes e registos de comparação desta nova ferramenta comparativamente com as já existentes.

1.4 Estrutura da Dissertação

Para além da introdução, esta dissertação contém mais 3 capítulos. No capítulo 2, é descrito o estado da arte e é apresentada toda a informação necessária ao leitor para que possa obter um total enquadramento na área de trabalho desta dissertação. É feita uma extensa descrição de todas as ferramentas, *software* e técnicas que atualmente são usadas, assim como explicação mais

pormenorizada do *software* a ser usado nesta dissertação No capítulo 3, procede-se à explicação de todas as escolhas feitas relativamente ao *software* no decorrer desta dissertação. Assim como apresenta a forma como a implementação foi realizada e os resultados que foram obtidos à medida que os testes foram sendo realizados. No capítulo 4 por fim, é feito um apanhado de toda a dissertação, frisando a verdadeira contribuição deste trabalho para a comunidade científica assim como a potencialidade do trabalho futuro.

Capítulo 2

Revisão Bibliográfica

Neste capítulo é descrito o estado da arte e são apresentados trabalhos relacionados para mostrar o que existe no mesmo domínio e quais os problemas em aberto. Deve deixar claro que existe uma oportunidade de desenvolvimento que cobre alguma falha.

Muitas empresas investem nos seus próprios centros de dados privados para satisfazer grande parte das suas necessidades a nível de alocação de dados. Contudo estes centros de dados, muitas vezes não são capazes de lidar com os grandes picos de fluxos de informação recorrentes numa rede de comunicação, levando desta forma a atrasos e a uma menor taxa de transferência. Usar poder computacional adicional para lidar com os já referidos imprevisíveis picos de informação é caro e tem-se revelando bastante ineficiente, tendo em conta que uma grande parte destes recursos não estarão a ser usados na maior parte do tempo. Recorrer à migração da aplicação na sua totalidade para a Cloud, embora fosse possivelmente mais barato do que investir num centro de dados privado, capaz de lidar com o contínuo fluxo de informação, tendo em conta que os servidores são só alugados quando necessário, seria ainda assim caro e iria comprometer a privacidade de informação importante da empresa. Será então considerada uma solução "ótima", recorrer a recursos adicionais na Cloud assim que os recursos do cluster local comecem a tornar-se lotados. Esta técnica é conhecida por Cloud Bursting e permite a uma empresa escalar, ainda que de forma virtual, a sua infraestrutura local "delegando" o processamento de algumas tarefas para uma Cloud pública assim que a necessidade se apresentar. Cloud Bursting pode então assim ser de grande utilidade para gestores de rede e especialistas em segurança, oferecendo-lhes desta forma uma grande capacidade computacional "temporária" sempre que necessário podendo efetuar detecções de intrusões e gestões de tráfego de aplicações mais complexas. Este processamento de informação do fluxo de uma rede de comunicação de uma empresa denomina-se por tráfego de rede.

A análise de tráfego de rede é importante na detecção de intrusões e na gestão de tráfego de aplicações. Soluções de análise de tráfego baseadas em clusters, de baixo de custo têm sido apresentadas para processamento em massa de grandes blocos de captura de tráfego, escalando a capacidade de processamento de um nó de análise de tráfego. Devido às variações da intensidade

de tráfego, causados pelos naturais picos de fluxo de informação, um cluster de análise de rede pode ter que ser aumentado de forma drástica para suportar a contínua captura e processamento de blocos de pacotes. "Delegar" a análise de alguns blocos de pacotes para a Cloud pode atenuar a necessidade de aumentar de forma drástica as dimensões do cluster local. É um facto que as soluções existentes para análise de tráfego de rede na Cloud já disponibilizam os benefícios dos serviços baseados em Cloud para os analistas de tráfego de rede e começam a "abrir algum caminho", para soluções de análise de tráfego Map-Reduce baseadas em Cloud. Supondo que um utilizador deseja processar dados que estão alocados localmente num supercomputador local. Considerando o caso que o utilizador precisa de analisar esta informação mas o recursos computacionais do supercomputador não estão disponíveis naquele preciso momento. Em vez de, enviar a tarefa com o bloco de informação de tráfego para ser analisado no supercomputador e esperar que os recursos computacionais estejam disponíveis, o utilizador pode preferir recorrer a um fornecedor de serviços públicos de Cloud. Considerando outra situação, onde um grupo de investigação alocou a sua informação num supercomputador local. Numa dada altura, o grupo de investigação pode desejar acrescentar nova informação sobre novas experiências ou simulações, e não existe espaço disponível para alocar essa mesma informação no supercomputador. Nesse caso pode ser preferível recorrer a espaço de armazenamento numa Cloud.

2.1 Análise de tráfego

Análise de tráfego é uma técnica de dedução que olha para padrões de comunicação entre entidades num sistema. Resume-se ao processo de intercetar e examinar mensagens de forma a deduzir informação a partir de padrões na comunicação. Pode ser executada até em mensagens que estão encriptadas e não podem ser desencriptadas. Na sua generalidade, quanto maior o número de mensagens analisadas, ou até intercetadas e armazenadas, então cada vez mais poderá inferir a partir do tráfego. Pode ser executada num contexto de inteligência militar, contra-inteligência, e é uma preocupação constante na área da segurança [FD10].

O tamanho dos pacotes que estão a ser trocadas entre dois *hosts* pode ser informação muito valiosa para alguém que pretenda extrair informação daquela comunicação, ainda que esta pessoa não seja capaz de visualizar o conteúdo do tráfego (estando ele encriptado ou simplesmente indisponível). Vendo uma pequena agitação de diferencial de pacotes de apenas 1 *byte* com pausas consistentes entre cada pacote poderá indicar uma sessão interativa entre dois *hosts*, em que cada pacote indica apenas uma *keystroke* [FD10] Grandes pacotes de informação transferidos ao longo do tempo indicam transferências de ficheiros entre *hosts*, indicando também que *host* está a enviar o ficheiro e que *host* está a receber o ficheiro.

Por si só esta informação pode não ser extremamente prejudicial para a segurança da rede mas um "atacante"criativo será capaz de combinar esta informação com outra, de forma a evitar mecanismos de segurança específicos.

2.2 Data Clustering

Clustering é a classificação sem supervisão de padrões (observações, itens de informação ou vetores de informação). O problema de *clustering* tem sido abordado em muitos contextos e por investigadores em muitas áreas. Isto reflete o seu amplo apelo e utilidade como um dos "passos"a percorrer no caminho da exploração da análise de dados. Contudo *Clustering* é um difícil problema de combinatória, e diferenças nas premissas e contextos nas diferentes comunidades têm dificultado a transferência de conceitos úteis e genéricos.

A análise de dados está na base de muitas aplicações de computação, seja na fase de *design*, or seja na sua parte de operações *online*. Os procedimentos de análise de informação podem ser dicotomizados em exploratório ou confirmatório, baseado na disponibilidade dos modelos apropriados para o *data source*, mas um elemento chave nos tipos de procedimento (quer formação de hipótese ou tomada de decisão) é o *grouping*, ou classificação das medições, baseado seja em (i) qualidade de ajuste a um determinado modelo, or (ii) agrupamentos naturais (*Clustering*) revelado durante a análise. Análise de Cluster é a organização da coleção de padrões (normalmente representados como um vetor de medições, ou um ponto num espaço multidimensional), em *Clusters* baseados em similaridade.

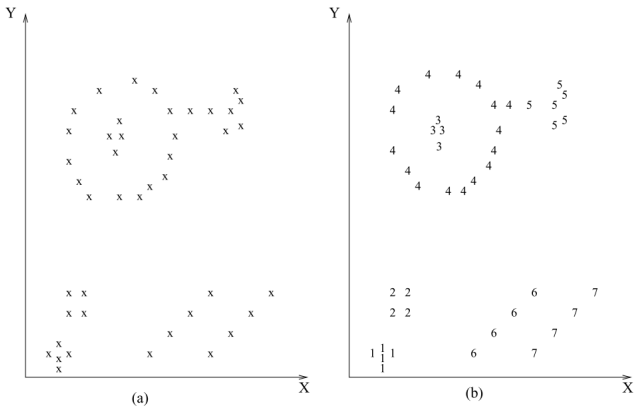

Figura 2.1: Data Clustering [JMF+00]

Intuitivamente, padrões com um *Cluster* válido são mais parecidos entre eles do que a um padrão que pertença a um *Cluster* diferente. Um exemplo de *Clustering* é retratado na figura 2.1. Os padrões de *input* são demonstrados do lado (a) da imgaem, e os *Clusters* desejados são demonstrados no lado (b) da figura. Aqui, aos pontos pertencentes ao mesmo *Cluster* é-lhes dada a mesma "classificação". A variedade de técnicas para representar informação, medição de proximidade (semelhança) entre elementos de informação, e agrupamentos de elementos de informação tem produzido uma seleção por vezes confusa de métodos de *Clustering*.

É importante perceber a diferença entre *Clustering* (classificação não supervisionada) e classificação "discriminante"(classificação supervisionada). Na classificação supervisionada, é-nos dado uma coleção de padrões previamente classificados. O problema jaz em classificar um padrão, acabado de encontrar que ainda não foi classificado. Tipicamente, os padrões classificados

(*Training*), são usados para aprender sobre a descrição das classes que, por sua vez, são usados para classificar um novo padrão. No caso de *Clustering*, o problema é agrupar uma dada coleção de padrões não classificados em *Clusters* com um significado. De certo modo, as classificações são associadas aos *Clusters*, mas estas classificações de categoria são orientadas à informação, isto é, estas são obtidas unica e exclusivamente a partir da informação.

Clustering é útil em várias análises de padrões exploratórias, agrupamentos, tomada de decisões, e situações de *Machine Learning*, inclusivamente *Data Mining*, segmentação de imagem e classificação de padrões. Contudo, em imensos problemas, existe um pouco de informação prévia (i.e. modelos estatísticos), disponível sobre os dados, e a tomada de decisão tem de assumir algumas premissas como válidas relativamente aos dados existentes.

As atividades de Clustering padrão envolvem os seguintes passos [JMF+00]:

1. Representação de padrões (podendo incluir extrações de *features* e/ou seleção).
2. Definição de uma medida de proximidade (semelhança) padrão para o domínio dos dados.
3. Clustering (*grouping*)
4. Abstração de dados (se necessário)
5. Avaliação do *output* (se necessário)

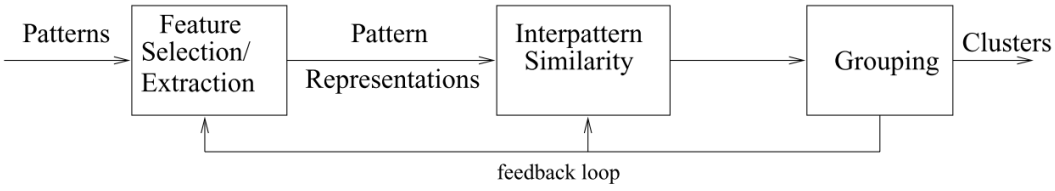

Figura 2.2: Stages of Clustering [JMF+00]

A figura 2.2 retrata uma típica sequência dos primeiros 3 destes referenciados passos, incluindo um caminho de *feedback* em que o resultado do processo de *grouping* poderá afetar a extração das *features* subsequentes e computação de semelhanças. A representação de padrões, refere-se ao número de classes, ao número de padrões disponíveis, e ao número, tipo e escala das *features* disponíveis para algoritmo de *Clustering*.

A seleção de *features* é o processo de identificar o subconjunto mais efetivo a partir das *features* originais a usar no processo de *Clustering*. A extração de *features* é o uso de uma ou mais transformações das *features* de entrada para produzir novas *features*. Qualquer uma das técnicas poderá ser usada para obter um conjunto de *features* a usar no processo de *Clustering*.

O padrão de semelhança é normalmente medido pela função de distância definida em pares de padrões. Uma variedade de medidas de distância nas mais diferentes comunidades [JMF+00]. Uma simples medida de distância como a distância Euclidiana pode muitas vezes ser usada para

refletir a diferença entre dois padrões, ao passo que outras medidas de semelhança podem ser usadas para caracterizar as semelhanças entre dois padrões. [JMF+00].

O passo de *grouping* pode ser realizado num grande número de maneiras diferentes. O resultado de *Clustering* pode ser difícil de distinguir (onde cada padrão tem um grau variável de "filiação"a cada um dos *Clusters* resultado. Algoritmos de *Clustering* Hierárquico produzem uma série de partições baseadas no critério para convergir ou separar *Clusters* baseados na semelhança. Algoritmos de *Partional Clustering* identificam a partição que otimiza (normalmente localmente), um critério de *Clustering*. Técnicas adicionais para a operação de *grouping*, incluem métodos de *Clustering* baseados em teoria de grafos e probabilidades [JMF+00].

A abstração de dados é o processo de extrair uma representação simples e compacta de um *dataset*. Aqui a simplicidade é da perspetiva da análise automática (de forma a que a máquina possa realizar processamento adicional de forma eficiente) ou é *human-oriented* (de forma a que a representação obtida seja de fácil compreensão intuitivamente atraente). No contexto de *Clustering*, uma abstração de dados comum é uma descrição compacta de cada *Cluster*, frequentemente em termos de protótipos de *Cluster* ou padrões representativos como um centróide.

Como é que o resultado de um algoritmo de *Clustering* é avaliado? Quais são as características que avaliam um "bom"resultado e um "mau"resultado?

Todos os algoritmos de *Clustering* irão produzir *Clusters* quando lhes são fornecidos dados como entrada - independentemente de se os dados possuem *Clusters* ou não. Se os dados não possuírem *Clusters*, alguns algoritmos de *Clustering* poderão obter "melhores"*Clusters* que outros. A avaliação do resultado de *Clustering*, então, possui várias "vertentes". Um é de facto a avaliação do domínio de dados em vez do algoritmo de *Clustering* por si só.

A existência de uma tão vasta coleção de algoritmos de *Clustering* na literatura desta área pode facilmente confundir um utilizador que se encontre a tentar selecionar um algoritmo que se ajuste ao problema em mãos. Um conjunto de critérios admissíveis por Fisher and Van Ness em 1971 são usados para comparar algoritmos de *Clustering* [JMF+00]. Estes critérios de admissibilidade são baseados em: (1) como os *Clusters* são formados, (2) a estrutura dos dados, e (3) a sensibilidade da técnica de *Clustering* para alterações que não afetam a estrutura de dados, contudo não existe uma análise importante de algoritmos de *Clustering* para lidar com questões importantes como:

- Como é que os dados deverão ser normalizados?
- Qual a medida de semelhança apropriada a usar numa dada situação?
- Como é que o conhecimento de domínio deverá ser usado num certo problema de *Clustering*
- Como é que um dataset muito grande (com milhões de padrões) deverá ser submetido a um *Clustering* apropriado?

Não existe nenhuma técnica de *Clustering* que seja aplicável de forma universal em revelar a variedade de estruturas presentes em *datasets* multidimensionais. Por exemplo, considerando um *dataset* de duas dimensões como na Figura 2.1. Nem todas as técnicas de *Clustering* pode revelar

todos os *Clusters* presentes com igual facilidade, dado que os algoritmos de *Clustering* possuem com regularidade premissas implícitas sobre a forma do *Cluster* ou configurações de mais de que um *Cluster* baseado nas medidas de semelhança e critérios de *grouping usados*.

Os seres humanos competem de forma aceitável com os procedimentos de *Clustering* automáticos em duas dimensões, mas a maior parte dos problemas reais envolvem *Clustering* de maiores dimensões. É difícil para os seres humanos obter uma interpretação intuitiva de dados incorporados no espaço de grandes dimensões. Em acréscimo, os dados raramente seguem o tipo de estruturas "ideal"como na Figura 2.1. Isto explica o grande número de algoritmos de *Clustering* que continuam a aparecer na literatura desta área. Cada novo algoritmo de *Clustering* apresenta uma ligeira melhoria na performance relativamente aqueles que já existem no que toca a uma distribuição de padrões específica.

É essencial para um utilizador, não só, possuir uma boa compreensão sobre a técnica em particular que será usada, mas também saber os detalhes do processo de angariamento de dados. Quanto mais informação o utilizador possuir sobre os dados em mãos, mais provável será que o utilizador seja capaz de ser bem sucedido a avaliar a verdadeira estrutura da classe de dados.

2.3 Algoritmos de *Clustering*

Organizar os dados em "grupos"com algum significado para o ser humano é das tarefas mais importantes e fundamentais para evoluirmos nesta área. Como exemplo poderemos ter, uma classificação científica em que dispõe os organismos em: Domínio, Reino, Filo, Classe, etc. A análise de *cluster* é o estudo formal de métodos e algoritmos para agrupar, ou realizar *clustering*, os objetos de estudo de acordo com as características intrínsecas ou analisadas ou semelhanças entre cada um dos casos. O objetivo de *clustering* é encontrar estrutura nos dados e é por consequente uma ciência exploratória. *Clustering* tem uma longa e rica história numa grande variedade de campos científicos. Um dos mais populares e simples algoritmos é o *K-Means* que foi publicado pela primeira vez em 1955.Apesar de o *K-Means* ter sido publicado há mais de 50 anos e desde então terem sido publicados milhares de algoritmos de *clustering*, este ainda é bastante usada pela comunidade científica [Jai10].

De uma forma geral, algoritmos de *clustering* podem ser divididos em 2 grupos: *hierarchical* and *partitional*. Os algoritmos de *Hierarchical clustering* encontram recursivamente *clusters* de forma aglomerativa (começando em cada ponto no seu próprio *cluster* e agrupando os *clusters* com maior nível de semelhança, sucessivamente formando assim uma hierarquia de *clusters*) ou então de forma divisiva (começando todos os pontos no mesmo *cluster* e recursivamente dividir cada *cluster* em *clusters* mais pequenos). Comparados aos algoritmos de *hierarchical clustering*, os algoritmos *partitional clustering* encontram todos os *clusters* em simultâneo, como uma partição dos dados e não impõem uma estrutura hierárquica. [WZL+07]

O input para um algoritmo *hierarchical* é uma matriz *n x n* em que *n* é o número de objetos que no qual o *clustering* será realizado.

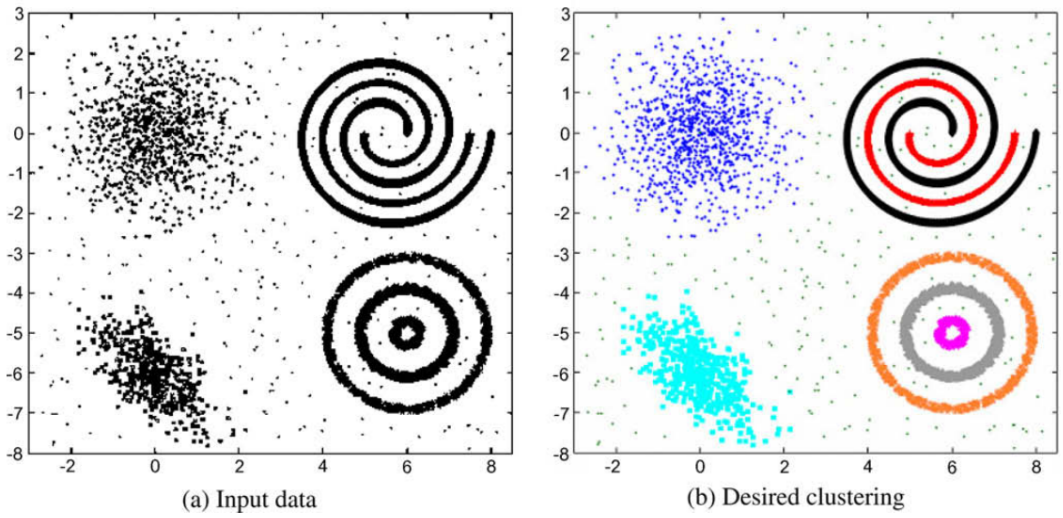

Figura 2.3: Exemplo de *Clustering* [Jai10]

Por outro lado um algoritmo *partitional* pode usar quer uma matriz *n x d* em que *n* objetos estão embutidos num espaço de *d* dimensões, quer uma matriz de *n x n*.

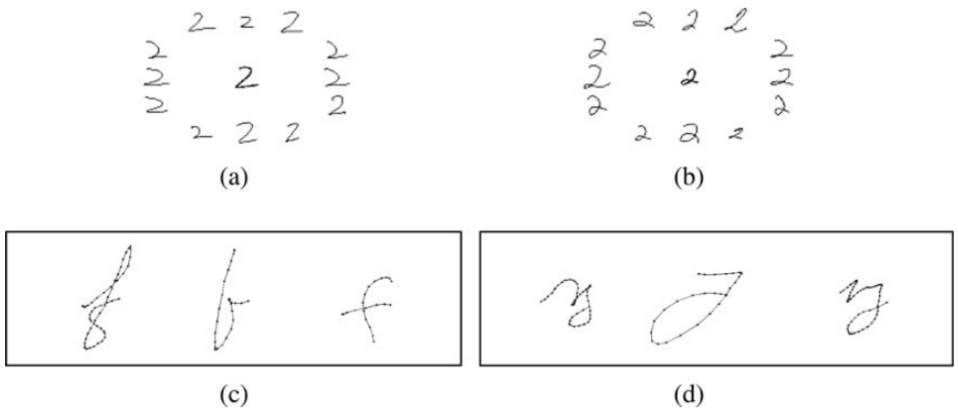

Figura 2.4: Exploração de nova Classe [Jai10]

Um exemplo de uma exploração de classe é demonstrado na Figura 2.4. Aqui o *clustering* foi usado para explorar novas subclasses numa aplicação de reconhecimento de caracteres manuscritos [Jai10]. Utilizadores diferentes escrevem os mesmos dígitos de diferentes maneiras, aumentando assim a variância dentro da própria classe. Executar *clustering* nos padrões de treino de uma classe pode descobrir novas subclasses, denominados lexemas nos caracteres manuscritos.

Em vez de ser usado um único modelo para cada caracter, múltiplos modelos baseados no número de subclasses são usados para melhorar a eficiência de reconhecimento.

Existe um grande leque de algoritmos para um utilizador que decida realizar *clustering*. Neste sentido que algoritmos de *clustering* devemos usar?

Como qualquer questão em *data science* e *machine learning* isso irá depender dos nossos dados. Um grande número de algoritmos é mais adequado para certas tarefas (como *co-clustering*, *bi-clustering*, ou realizar *clustering* em *features* em vez de *data points*). Claramente um algoritmo especializado em *text clustering*, irá ser a escolha óbvia para realizar *clustering* em *text data*, e outros algoritmos especializados em outro tipo específico de dados.

Logo, se soubermos o suficiente sobre os nossos dados, podemos restringir a procura ao algoritmo de *clustering* que mais se adequa à informação em mãos, ou aos tipos de propriedades importantes que a nossa informação possui, ou ao tipo de *clustering* específico que precisamos de realizar.

2.3.1 K-Means

K-Means é o algoritmo de eleição para muito simplesmente porque é rápido, fácil de compreender, e disponível em todo o lado (existe uma implementação em praticamente qualquer ferramenta de estatística ou *machine learning* que se use). Contudo o *K-Means* apresenta alguns problemas. O primeiro é que na sua génese não é propriamente um algoritmo de *clustering*, mas sim um *partitional clustering*. Com isso pretende-se dizer que o *K-Means*, não "procura"*clusters*, ele particiona o *dataset* inicial em vários "pedaços". Isso leva ao segundo problema: é necessário especificar exatamente à partida quantos *clusters* são esperados no final do processo. Se existir informação suficiente sobre os nossos dados, então provavelmente isso é algo que não se tornará num grande problema. Se, por outro lado, estiver simplesmente a explorar um novo *dataset* então o "número de *clusters*"é parâmetro díficil de "adivinhar". A solução tradicional é executar o K-Means várias vezes com um "número diferente de *clusters*"e tentar encontrar o valor mais adequado. Por fim o *K-Means* também está dependente da inicialização que poderá influenciar o resultado final. Fornecendo-lhe múltiplas inicializações diferentes e aleatórias pode levar à geração de *clusters* bem diferentes. Isto não gera muita confiança em qualquer *cluster* que possa produzido.

K-Means começa com uma partição inicial com *K clusters* e atribui padrões aos *clusters* de forma a reduzir o *squared error*. Dado que o *squared error* diminui sempre com um aumento no número de *clusters K*, pode apenas ser minimizado para um número fixo de *clusters*. Os principais passos para o algoritmo *K-Means* são os seguintes [Jai10]:

1. Selecciona uma partição inicia com *K clusters*. Repetir os passos 2 e 3 até os *clusters* estabilizarem.

2. Gerar uma nova partição atribuindo um novo padrão ao centro *cluster* mais próximo.

3. Calcular novos centros de *clusters*.

O algoritmo *K-Means* tem como requisito inicial três parâmetros especificados pelo utilizador: número de *clusters K*, inicialização de *clusters*, a métrica de distância. A escolha mais complica usualmente reside no número *K*. Tendo em conta que não existe nenhum critério matemático perfeito, um número de heurísticas estão disponíveis para a escolha de *K*. Tipicamente, o *K-Means*

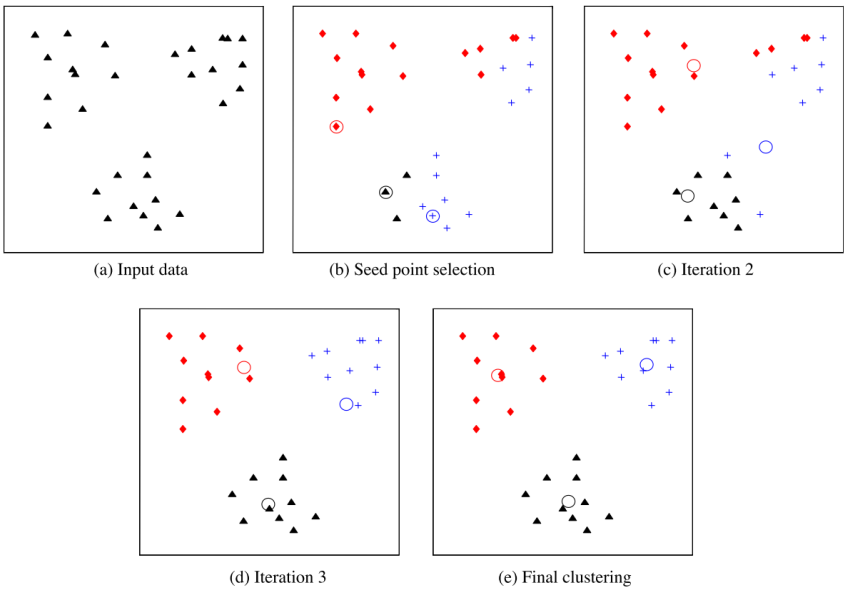

Figura 2.5: Ilustração de algoritmo *K-Means*. (a) Dados de *input* de 2 dimensões com 3 *clusters*; (b) Três pontos de "semente"selecionados como centros de *cluster* e atribuição inicial dos pontos aos *clusters*;(c) e (d) iterações intermédias a atualizar as etiquetas dos *clusters* e os respetivos centros ;(e) *clustering* final obtido a partir do *K-Means* em convergência. [Jai10]

"corre"independentemente para diferentes valores de K e a partição que parece ser mais significativa para o domínio é seleccionada. Inicializações pode levar a *clusterings* finais diferentes pois o *K-Means* só consegue convergir para um mínimo local. Uma maneira de ultrapassar o mínimo local é correr o algoritmo *K-Means*, para um dado K, com múltiplas partições iniciais diferentes, e escolher a partição com o menor *squared error*. O *K-Means* é tipicamente usado com a métrica Euclidiana para calcular a distância entre os pontos e os centros dos *clusters*. Como resultado, o *K-Means* encontra *clusters* esféricos ou forma de "bola"nos dados.

2.3.2 Affinity Propagation

[XWZZ08] [WZL+07]

O algoritmo *Affinity Propagation* tem como entrada uma coleção de valores reais que representam semelhanças entre pontos em que a semelhança s(i,k) indica o quão bom o ponto com index k é, para ser o centro da classe do ponto i. Quando o objetivo é minimizar o *squarred error*, cada semelhança é uma distância Euclidiana negativa: para pontos x_i e x_k, s(i,k)= - $\|x_i - x_k\|^2$.

Em vez de precisar do número de *clusters* especificados, o algoritmo *Affinity Propagation* leva como input um número real s(k,k) para cada ponto k de forma a que os pontos com maiores valores de s(k,k) sejam, com maior probabilidade, escolhidos para centros de classe. Estes valores são referido como "preferências". O *Affinity Propagation* pode ser visto como procurar configurações válidas das etiquetas $c = c_1, c_2, c_3, ..., c_n$ para minimizar a energia:

$$E(c) = -\sum_{i=1}^{N} s(i,\ c_i).$$

Figura 2.6: Minimizar a energia [XWZZ08]

O processamento do *Affinity Propagation* pode ser visto como um processo de comunicação de mensagens num grafo de fatores.[XWZZ08] Há dois tipos de mensagens trocadas entre pontos, por exemplo, "responsabilidade"e disponibilidade". A responsabilidade r(i,k), enviado do ponto *i*, reflete a prova de como o ponto *k* serve como exemplar do ponto *i*, tendo em conta outros potenciais exemplares para o ponto *i*. A disponibilidade de a(i,k), enviado do candidato exemplar *k* para o ponto *i*, reflete a prova acumulado de como o ponto *i* deveria escolher o ponto *k* como seu exemplar, tendo em conta o apoio de outros pontos que acham que ponto *k* deveria ser um exemplar.

As mensagens têm que ser trocadas entre pares de pontos com semelhanças conhecidas.

Quanto aos passos do algoritmo em si, são os seguintes [XWZZ08]:

Input: s(i, k): a semelhança entre o ponto *i* e o ponto *k*. p(j): o *array* de preferências que indica a preferência para que o ponto *j* seja o centro do *cluster*. Output: idx(j): o índice do centro do *cluster* para o ponto *j*. dpsim: a soma das semelhanças dos pontos para os centros de cada um dos seus *clusters*. expref: a soma das preferências dos centros de *clusters* identificados. netsim: soma das semelhanças e das preferências dos pontos.

Passo 1: Inicializa as disponibilidades a(i,k) a 0:

$$a(i,\ k)=0.$$

Figura 2.7: Passo 1 do *Affinity Propagation* [XWZZ08]

Passo 2: Atualiza as responsabilidades usando a regra:

$$r(i,k) \leftarrow s(i,k) - \max_{k' \text{ s.t. } k' \neq k} \{a(i,k') + s(i,k')\}.$$

Figura 2.8: Passo 2 do *Affinity Propagation* [XWZZ08]

Passo 3: Atualiza a disponibilidade usando a regra:

$$a(i,k) \leftarrow \min\left\{0,\ r(k,k) + \sum_{i' \text{ s.t. } i' \neq i,k} \max\{0,\ r(i',k)\}\right\}.$$

Figura 2.9: Passo 3 do *Affinity Propagation* [XWZZ08]

Passo 4: A própria disponibilidade é atualizada de forma diferente:

$$a(k,k) \leftarrow \sum_{i' \text{ s.t. } i' \neq k} \max\{0, r(i',k)\}.$$

Figura 2.10: Passo 4 do *Affinity Propagation* [XWZZ08]

2.3.3 Mean Shift

A ideia principal do algoritmo *Mean Shift* é tratar os pontos num espaço de d dimensão como uma função empírica probabilística de densidade em que regiões de densidade correspondem ao máximo local. Para cada ponto no espaço, um realiza um procedimento de gradiente crescente na densidade até atingir a convergência. Os pontos estacionários deste processo representam os modos de distribuição. Os pontos associados com o mesmo ponto estacionário são considerados pontos do mesmo *cluster*.

O procedimento do algoritmo *Mean Shift* para um dado ponto x_i é o seguinte [Der05]:

1. Calcular o vetor *Mean Shift* $m(x_i^t)$
2. Transladar a janela de densidade: $x_i^{t+1} = x_i^t + m(x_i^t)$
3. Iterar entre os passos 1 e 2 até atingir a convergência. $\Delta f(x_i)$

O componente de mais "pesado"a nível de computação no processo *Mean Shift* é a identicação dos "vizinhos"de um ponto no espaço. Esta computação torna-se demasiado complicada para espaços com dimensões muito grandes.

2.4 Tensorflow

Tensorflow é uma interface para expressar algoritmos de *machine learning*, e uma implementação para executar esses algoritmos. A computação expressada usando o *Tensorflow* pode ser executada com muito pouca diferença de código numa grande variedade de sistemas heterogéneos, desde dispositivos móveis como telemóveis e *tablets* até sistemas distribuídos de grande escala de centenas de máquinas e milhares de dispositivos de computação como GPU's. O sistema é flexível e pode ser usado para expressar uma grande variedade de algoritmos, incluindo *training* e inferência de algoritmos para modelos de redes neuronais, e tem sido usado para realizar pesquisa e para o desenvolvimento sistemas de *machine learning* para a produção em mais de uma dúzia de áreas de ciência da computação e outros campos de pesquisa, como reconhecimento de discurso, visão computacional, robótica, obtenção de informação, processamento de linguagem natural, extração de informação geográfica.

Uma computação em *Tensorflow* é descrita como um grafo dirigido, que é composto por um conjunto de nós. O grafo representa uma computação de fluxo de dados, com extensões para permitir que alguns tipos de nós mantenham e atualizem o *persistent state* e para ramificar e iterar estruturas de controlo dentro do grafo. Os clientes tipicamente constroem um grafo computacional usando um das linguagens de *frontend* (C++ ou Python) [AAB+15]. Um exemplo de um pedaço de código a construir e a ser depois executado é mostrado na Figura 2.11 o grafo de computação como resultado é mostrado na Figura 2.12.

```
import tensorflow as tf

b = tf.Variable(tf.zeros([100]))                    # 100-d vector, init to zeroes
W = tf.Variable(tf.random_uniform([784,100],-1,1)) # 784x100 matrix w/rnd vals
x = tf.placeholder(name="x")                        # Placeholder for input
relu = tf.nn.relu(tf.matmul(W, x) + b)              # Relu(Wx+b)
C = [...]                                           # Cost computed as a function
                                                    # of Relu

s = tf.Session()
for step in xrange(0, 10):
  input = ...construct 100-D input array ...        # Create 100-d vector for input
  result = s.run(C, feed_dict={x: input})           # Fetch cost, feeding x=input
  print step, result
```

Figura 2.11: Exemplo de pedaço de código em *Tensorflow* [AAB+15]

Num grafo de *Tensorflow*, cada nó tem zero ou mais dados de entrada e zero ou mais dados de saída, e representa a instanciação de uma operação. Os valores que fluem ao longa das arestas no grafo, são *tensors*, *arrays* de dimensão arbitrária em que os elementos subjacents são especificados ou inferidos no tempo de construções do grafo. Arestas especiais, chamadas dependências de controlo, também podem existir no grafo: não existe fluxo de dados ao longo dessas arestas, mas indicam que o nó "fonte"para o controlo de dependências tem de terminar de executar antes do nó de destino para que a dependência de controlo comece a executar.

Em grande parte das computações o grafo é executado várias vezes. A maior parte dos *tensors*, não sobrevive à primeira execução. Contudo, uma Variável é um tipo especial de operação que retorna um *handle*, para um *tensor* mutável que sobrevive ao longo das execuções do grafo. *Handles* para estes *tensors* mutáveis podem ser usados para várias operações especiais, como *Assign* e *AssignAdd* (equivalente a +=), que modificam o *tensor* referido. Para aplicações de *machine learning* em *Tensorflow*, os parâmetros do modelo são tipicamente armazenados em *tensors*, mantidos em variáveis, e são atualizados em part do *Run*, do grafo de *training* do modelo.

Os componentes principais num sistema de *Tensorflow* são o *Client*, que usa a interface da *Session* para comunicar com o *Master*, e com um ou mais *worker processes*, com cada *worker process* responsável por arbitrar o acesso para um ou mais dispositivos de computação (como

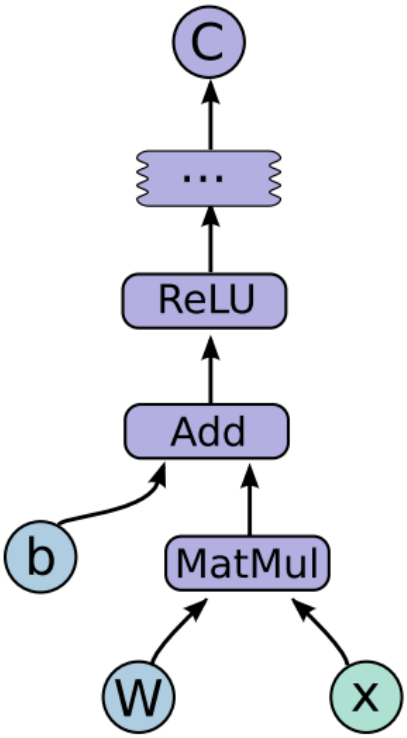

Figura 2.12: Grafo de computação referente ao código da figura 2.11. [AAB+15]

núcleos CPU ou GPU) e por executar os nós dos grafos nesses mesmos dispositivos como foram instruídos pelo *Master*.

Existem implementações tanto locais como distribuídas para a interface de *Tensorflow*. A implementação local é usada quando o *Client*, o *Master*, e o *Worker* correm na mesma máquina no contexto de apenas um processo de sistema operativo (com possivelmente múltiplos dispositivos, se por exemplo, a máquina possuir mais do que uma GPU). A implementação distribuída, partilha a maior parte do código com a implementação local, mas extende com recorrência a um ambiente externo em que o *Client*, o *Master*, e os *workers*, podem estar em processos diferentes em máquinas diferentes. No sistema distribuído, as diferentes tarefas são contentores nos trabalhos geridos por um sistema de *scheduling cluster*. Estes dois modos diferentes estão ilustrados na Figura. Dispositivos são o núcleo computacional do *Tensorflow*. Cada *worker* é responsável para um ou mais dispositivos, e cada dispositivos tem um tipo de dispositivo, e um nome. Os nomes de dispositivos são compostos por pedaços que identificam o tipo de dispositivo, os índice do dispositivo dentro do *worker*, e na opção distribuída, uma identificaçãodo trabalho e tarefa do *worker*. Nomes de dispositivo de exemplo são "/job:localhost/device:cpu:0"ou "/job:worker/task:17/device:gpu:3". Existem implementações da interface de dispositivos para CPU's e GPU's, e implementações para outros dispositivos que podem ser fornecidos através de um mecanismo de registo. Cada objeto de dispositivo é responsável por gerir a alocação e desalojamento da memória do dispositivo, e

organizar tudo para a execução de qualquer *kernel* que se seja necessária por níveis mais altos da implementação do *Tensorflow*.

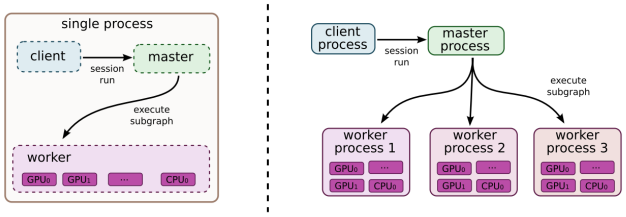

Figura 2.13: Estrutura de uma máquina e de sistema distribuído. [AAB+15]

2.5 Outras Ferramentas

2.5.1 *MATLAB*

MATLAB trata-se de um software interativo de alta performance orientado para o cálculo numérico. *MATLAB* integra análise numérica, cálculo com matrizes, processamento de sinais e construção de gráficos num ambiente de fácil uso onde os problemas e soluções são expressos apenas como são escritos de forma matemática, ao contrário da programação tradicional.

O *MATLAB* é um sistema interativo cujo elemento básico de informação é uma matriz. Esse sistema permite a resolução de muitos problemas numéricos com apenas uma fração do tempo que se gastaria para escrever um programa semelhante numa qualquer outra linguagem de programação. Além disso, as soluções dos problemas são expressas quase exatamente como elas são escritas de forma matemática.

MATLAB é um "Matrix Laboratory"e, como tal, fornece muitas maneiras convenientes para a criação de matrizes de várias dimensões. No vocabulário *MATLAB*, um vetor refere-se a uma dimensão (1 x N ou N x 1), de forma tradicional referida como um vetor em outras linguagens de programação. A matriz geralmente refere-se a uma matriz multidimensional, isto é, uma matriz com mais de uma dimensão, por exemplo, uma N x M, uma N x M x L, etc, onde N, M, L, são superiores a 1. Em outras linguagens, uma matriz pode ser referida como um array de arrays, ou array de arrays de arrays, ou simplesmente como um array multidimensional.

2.5.2 *Python*

Python é uma linguagem de programação dinâmica, bastante usada, de uso genérico, de *high-level*. A sua filosofia de design enfatiza a perceptibilidade de código, e a sua síntaxe permite aos programadores exprimir conceitos em menos linhas de código que o possível que linguagens como C++ ou Java. A linguagem providencia construtores intencionados a permitir programas "limpos"em pequena e larga escala.

Python permite múltimos paradigmas de programação, inclusivamente orientados a objetos, e programação funcional ou estilos processuais. Permite um sistema de tipo dinâmico e gestão de memória automática e tem uma grande e compreensiva biblioteca *standard*.

Os intérpretes de *Python* estão disponíveis para muitos sistemas operativos, permitindo ao código de *Python* correr numa grande variedade de sistemas. Usando ferramentas de terceira parte, como *Py2exe* ou *Pyinstaller*, o código de *Python* pode ser "empacotado"em programas executáveis independentes para alguns dos sistemas operativos mais populares, por isso o *software* baseado em *Python* pode ser distribuído, usado, nesses ambientes sem necessidade de instalar um intérprete de *Python*.

Scikit-learn providencia um alcance de algoritmos de aprendizagem supervisionados e não supervisionados através de uma interface consistente em *Python*. É autorizado através de uma licença *BSD* permissiva simplificada é distribuída através de de muitas distribuições de *Linux*, encorajando desta forma o uso académico e comercial.

A biblioteca é construída sobre o *SciPy (Scientific Python)* que tem de ser instalado antes que se possa usar o *scikit-learn*. Esta grande quantidade inclui:

- *NumPy: Base n-dimensional array package.*
- *SciPy: Fundamental library for scientific computing.*
- *Matplotlib: Comprehensive 2D/3D plotting.*
- *IPython: Enhanced interactive console.*
- *Sympy: Symbolic mathematics.*
- *Pandas: Data structures and analysis.*

As extensões e módulos para *SciPy* tradicionalmente são chamados *SciKits*. Deste modo, o módulo providencia algoritmos de aprendizagem é chamado *scikit-learn*.

A visão para a biblioteca é um nível de robustez e apoio necessário para uso em sistemas de produção. Isto traduz-se num grande foco no que toca a facilidade de uso, qualidade de código, colaboração, documentação e performance.

Apesar de a interface ser em *Python*, bibliotecas em C são o que potenciam a performance em *numpy* para *arrays* e operações em matrizes.

2.5.3 *R*

R é um conjunto de capacidades de *software* para manipulação de dados, cálculo e exposição gráfica de informação. Entre outras coisas, *R* possui a habilidade para lidar com informação de forma eficiente e armazenamento de informação, um conjunto de operadores para executar cálculos em *arrays*, em particular em matrizes, uma grande, coerente e integrada coleção de ferramentas intermediárias para análise de dados, habilidade de exposição de dados para análise de dados quer

diretamente no computador quer em cópia impressa, e um linguagem de programação efetiva, simples e bem desenvolvida (denominada "S"), que inclui condicionais, *loops*, funções recursivas definidas pelo utilizador e capacidade de *input* e *output*. (De facto a maior parte das funções providenciadas pelo sistemas são elas próprias escritas na linguagem "S). O termo "ambiente"é suposto caracteriza-lo como um sistema coerente e perfeitamente planeado, em vez dum acréscimo de ferramentas muito específicas e inflexíveis, como muito tradicionalmente se encontra com outro *software* de análise de dados.

R é então um "veículo"para métodos desenvolvidos recentemente de análise de dados interativa. Desenvolveu-se bastante rápido, e tem sido estendido por uma larga coleção de pacotes. Contudo, a maior parte dos programas escritos em *R* são essencialmente efémeros, escritos apenas para uma análise de dados muito específica.

2.6 Machine Learning

Machine Learning é um método de análise de dados que automatiza a construção de modelos analíticos. Usando algoritmos que iterativamente "aprendem"a partir dos dados, *machine learning* permite aos computadores, descobrir e compreender sem terem que ser especificamente programados para onde olhar.

O aspeto iterativo de *machine learning* é importante na medida em que os modelos são expostos para nova informação, e é assim possível que eles se adaptem independentemente. Eles aprendem a partir de computações prévias para produzir decisões e resultados confiáveis e repetíveis. É uma ciência que não é nova de todo, mas ainda assim tem vindo a ganhar imensa importância ao longo dos últimos anos.

Devido às novas tecnologias de computação, *machine learning* de hoje em dia não é igual ao que era no passado. Ao passo que muitos algoritmos de *machine learning* já foram criados há bastante tempo, a capacidade para executar cálculos matemáticos complexos de forma automática em grandes quantidades de informação, vezes e vezes sem conta, cada vez mais rápido, é um desenvolvimento relativamente recente. Temos como exemplos muito práticos desta mesma tecnologia:

- O bastante conhecido, carro da Google que é capaz andar na rua sem condutor. Representa a própria essência de *machine learning*.
- Recomendações *online* de ofertas como aquelas na Amazon ou Netflix. São aplicações de *machine learning* para o dia-a-dia.
- Saber o que consumidores postam no Twitter sobre uma determinada marca. É uma combinação de *machine learning* com criação de regras linguísticas.
- Deteção de fraudes. É uma das mais óbvias e importantes aplicações de *machine learning* nos dias de hoje.

O ressurgente interesse em *machine learning* é devido aos mesmo fatores que tornaram *data mining* ainda mais popular. Coisas como o crescente volume e aumento na variedade de dados disponíveis, processamento computacional cada vez mais barato e poderoso e armazenamento de dados mais barato. Todas estas cpisas significam que é possível produzir modelos mais rápida e automaticamente que possam analisar, dados em maiores dimensões e com maior nível de complexidade e chegar a resultados mais rapidamente de forma mais precisa, mesmo em grande escala. Produzindo assim previsões que nos possam guiar a melhores decisões e tomada de ações mais inteligentes em *real time* sem intervenção de seres humanos.

Uma das chaves para produzir ações inteligentes em *real time* é automatizar a construção de modelos.

2.7 Resumo ou Conclusões

Esta extensa revisão bibliográfica forneceu as bases para que essencialmente pudessem ser tomadas decisões de escolha de *software*, preferência de algoritmos, entre outros, de uma forma mais informada. Foi também necessário realizar imenso estudo de antes da realização do trabalho mais especificamente sobre o *Tensorflow*, para que pudesse perceber essencialmente as suas limitações e qual o seu verdadeiro propósito aquando da altura de concepção.

Capítulo 3

Implementação

Neste capítulo será explicado o porquê da escolha dos algoritmos de *clustering* usados, assim como as razões que levaram à tomada de algumas decisões aquando da Implementação. Posteriormente irão ser explicados os passos tomados na questão da implementação dos algoritmos propriamente dita.

3.1 Algoritmos

Como já referido anteriormente, existe uma enorme variedade de algoritmos de *clustering*, e como tal a tarefa de escolher quais os algoritmos a usar torna-se bastante complexa. Nesse sentido temos de establecer critérios e prioridades de forma a podermos escolher alguns de entre todos os existentes.

Um dos principais critérios que levou à escolha dos algoritmos foi o tipo de dados em mãos. Tendo em conta que alguns dos datasets usados requeriam o uso de muitos clusters para podermos obter bons resultados, neste sentido temos de excluir algoritmos como *Spectral clustering*, que diminui significativamente a sua performance quando o número de *clusters* se torna demasiado elevado, assim como, o *Hierarchical clustering*, que se torna demasiado dispendioso quando os *datasets* se tornam muito grandes.

DBSCAN foi também excluído pelo facto de apenas produzir resultados conclusivos com *datasets* que possuam alta densidade, ao passo que "exclui"pontos de *datasets* que sejam mais dispersos levando assim à perda de informação significativa.

Por fim foi excluído o algoritmo *Agglomerative Clustering*, pois seria um dos algoritmos que iria consumir mais tempos de implementação, pondo assim em risco a implementação na sua totalidade. Foi então decidido deixar a implementação deste último algoritmo para último lugar. Infelizmente devido ao tempo despendido a correções e acertos nos restantes algoritmos, não foi possível chegar a uma implementação estável deste mesmo algoritmo.

3.2 Tensorflow

Num sistema comum, existem múltiplos dispositivos computacionais. No Tensorflow os dispositivos suportados são CPU e GPU. Permite dividir as tarefas de um dado programa pelos diferentes CPU's/GPU's disponíveis, aumentando imenso assim a sua performance.

- `"/cpu:0"`: The CPU of your machine.
- `"/gpu:0"`: The GPU of your machine, if you have one.
- `"/gpu:1"`: The second GPU of your machine, etc.

Figura 3.1: Possibilidades de CPU e GPU para Tensorflow. [Cpu]

Um dos grandes entraves desta ferramenta é que as únicas GPU's que o Tensorflow é capaz de suportar são de arquitectura NVIDIA. Isto faz com que qualquer máquina que não possua um processador gráfico NVIDIA, apenas possa utilizar o Tensorflow recorrendo ao módulo de CPU's, o que francamente, reduz significativamente o poder de processamento desta ferramenta.

Neste sentido foi necessário recorrer a uma máquina fornecida pela FEUP com um processador gráfico GeForce GT 720 de forma a poder utilizar o Tensorflow recorrendo a esta vertente. Todos os testes realizados no decorrer da implementação, foram executados com recurso a esta GPU apenas.

Desta maneira foi então possível encontrar uma forma estável de poder executar os já referidos algoritmos recorrendo diferentes ferramentas numa só máquina para que se possa efetivamente realizar testes e obter um termo válido de comparação.

3.3 Implementação de algoritmos

Um dos grandes entraves que foi encontrado aquando da implementação foi sem dúvida desenvolver uma forma de implementar os algoritmos de forma igualmente eficiente de forma a que não houve alterações de performance nas diferentes ferramentas para que se pudesse então realizar comparações.

Foi também necessário encontrar *datasets* de referência para que todos os testes realizados possam replicados se necessário. Recorreu-se então inicialmente a um *dataset* bastante conhecido na área do *clustering*: *Iris dataset*.

Foi escolhido esse mesmo *dataset* pois enquadra-se bastante bem no problema em mãos tendo em conta a sua dimensão e número de *features*.

A implementação dos algoritmos foi feita de forma gradual tendo tido a seguinte ordem:

1. *K-Means*

2. *Affinity Propagation*

3. *Mean Shift*

Não houve um critério de selecção para a implementação dos algoritmos, sendo que o *K-Means* foi o primeiro a ser implementado simplesmente por ser um algoritmo com o qual já tinha alguma significativa familiaridade.

Apesar de já existirem várias implementações do algoritmo para as diferentes ferramentas *K-Means* nenhuma delas estava especificamente orientada à performance do algoritmo, nem permitia qualquer tipo de alteração nas configurações do algoritmo. Nesse sentido foi necessário reescrever inteiramente o algoritmo aproveitando apenas alguns traços e ideias gerais que já existia sobre como implementar de forma eficiente os 3 algoritmos já referidos.

Em título de exemplo no algoritmo *K-Means* grande parte das implementações existentes, não possuíam qualquer tipo de condição de paragem, limitando-se apenas a realizar um número, já predefinido em antemão, de iterações. E com a implementação realizada foi possível acrescentar uma condição válida de paragem. Em cada iteração o centro dos *clusters*, é atualizado tendo em conta os pontos atribuídos aquele *cluster*. Foi então agora possível terminar o algoritmo quando o centro dos *clusters* não são atualizados e por conseguinte alcançamos os pontos ótimos como centros dos *clusters*.

3.3.1 *K-Means*

Apesar de todas as plataformas alcançarem os mesmo resultados (os mesmos *clusters* para os mesmos *datasets*), verificaram-se significativas diferenças em termos de performance.

Com um *dataset* com 150 entradas, o *Tensorflow* foi capaz de executar toda a computação com o algorimo *K-Means* e encontrar os *clusters* como pode ser visualizado na Figura 3.2 em 1.9 segundos, ao passo que o Matlab demorou 3.9 segundos para executar essa mesma computação como é vísivel na Figura 3.3 ao passo que o R conseguiu realizar esta mesmo computação em 4.2 segundos e em Python foi possível chegar aos resultados em 3.5 segundos.

Com estes resultados podemos então concluir que para um *dataset* de dimensão relativamente pequena se observa uma ligeira melhoria de performance do *Tensorflow* relativamente às outras ferramentas.

Com um *dataset* com 434874 entradas a diferença torna-se então bastante significativa tendo o *Tensorflow* calculado os *clusters* em 394 segundos tendo conseguido a seguinte demonstração como vísivel na Figura 3.5. Para este mesmo *dataset* o *MATLAB* demorou 934 segundos, ao passo que *Python* demorou 813 segundos e *R* demorou 690 segundos.

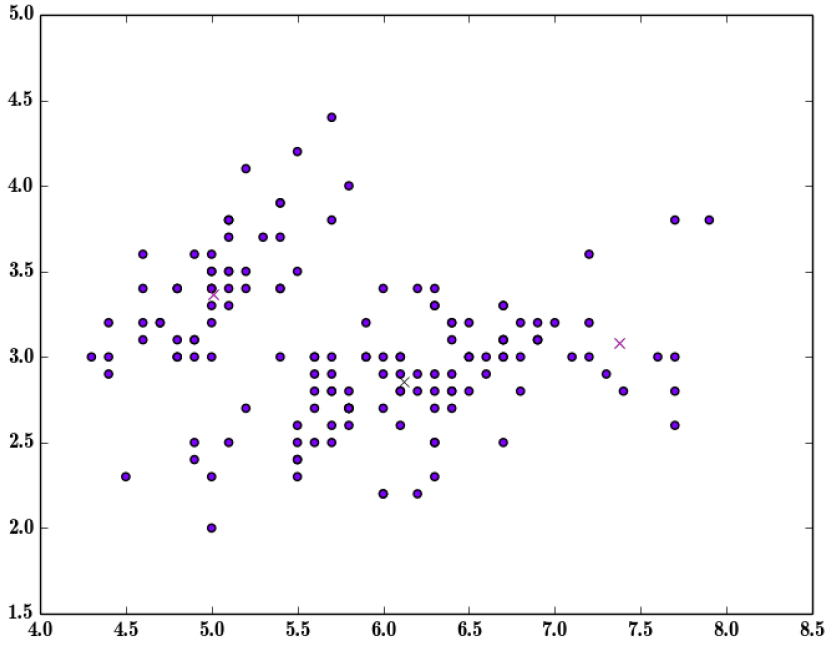

Figura 3.2: *K-Means -> Tensorflow -> dataset* de 150 entradas.

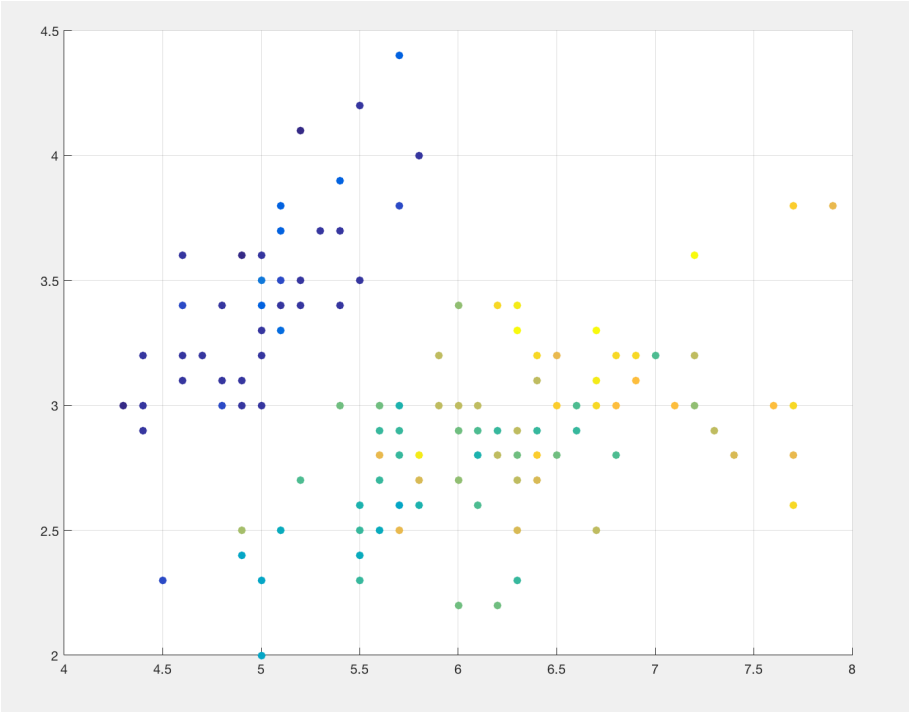

Figura 3.3: *K-Means -> Matlab -> dataset* de 150 entradas.

3.3.2 *Affinity Propagation*

Quanto a este algoritmo, como já referido anteriormente, liberta-nos de termos que especificar em antemão o número de *clusters* como no caso do *K-Means*, ainda assim é necessário fornecer

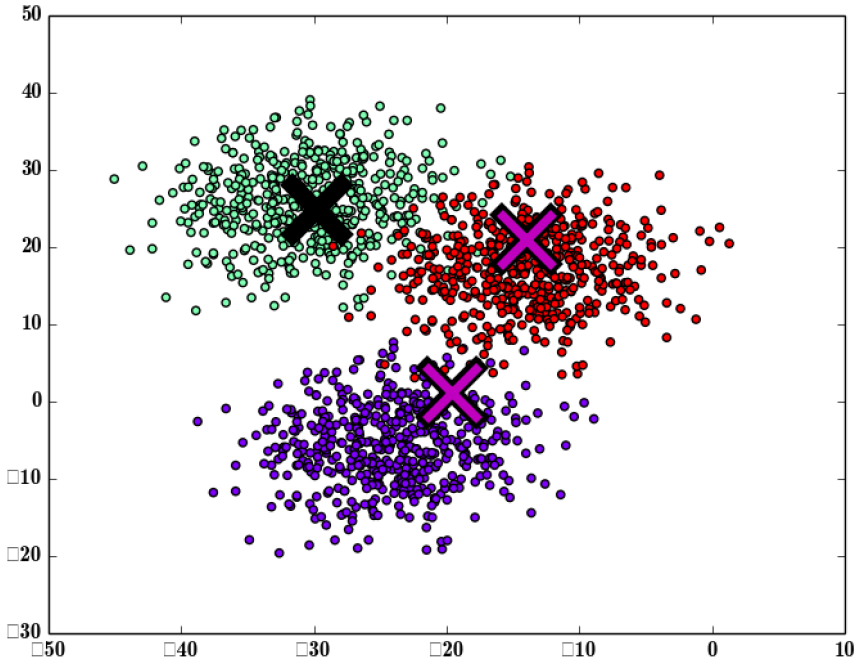

Figura 3.4: *K-Means -> Tensorflow -> dataset* de 434874 entradas.

variáveis que irão afetar a performance do algoritmo: *preference* e *damping*. Para um qualquer utilizador que não possui qualquer tipo de experiência em algoritmos de *clustering*, muito menos com este algoritmo em específico, é necessário despender algum tempo em experiências, de forma a ser claro de que forma estes valores afetam a performance do algoritmo.

O *Affinity Propagation* identifica-se por ser um algoritmo bastante fiável e produz resultados bastante válidos e consistentes mas é relativamente lento.

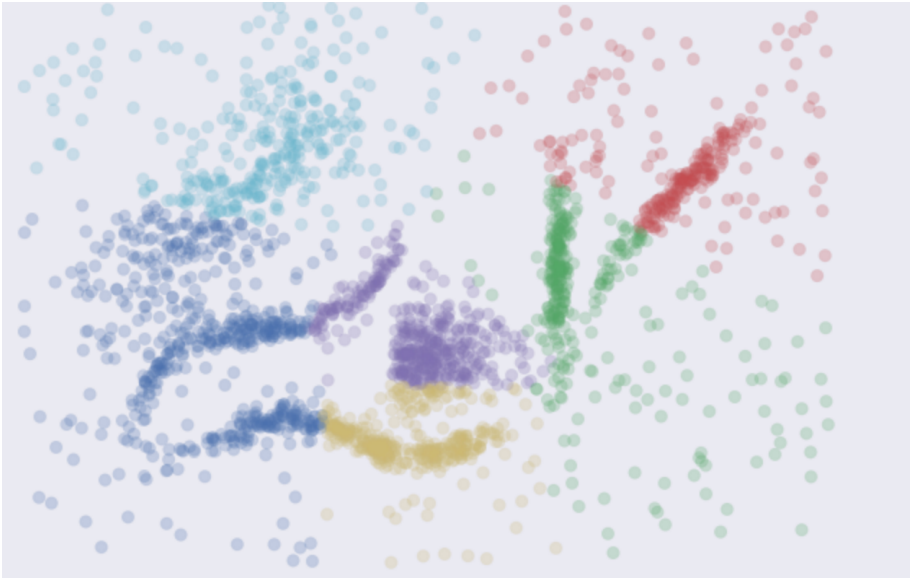

Figura 3.5: *Affinity Propagation -> Python*

Como tal podemos verificar que o *Tensorflow* executa este algoritmo com uma média de 2.2 segundos, para um *dataset* de 150 entradas, ao passo que o *Matlab* e o *R* demoraram aproximadamente 3.9 segundos. Por último, este algoritmo implementado em *Python* demorou aproximadamente 4.2 segundos até serem alcançados os *clusters* finais.

Com um *dataset* de maiores dimensões a falta de velocidade foi ainda mais acentuada sendo que no *Tensorflow* para um *dataset* de 434874 entradas demorou 510 segundos, que neste caso ainda representa uma diferença significativa em comparação com *K-Means*. *MATLAB* demorou 1256 segundos, *Python* demorou 1132 segundos e *R* demorou 980 segundos.

3.3.3 *Mean Shift*

O algoritmo de *Mean Shift* por sua vez, é lento mas permite uma muito boa escalabilidade sendo que por definição este algoritmo não tem em consideração todos os pontos, sendo que dá prioridade às zonas de grande densidade de pontos no espaço.

Usando um *dataset* de 150 entradas no *Tensorflow* foi possível obter o resultado ao final de 3.3 segundos, 4.1 segundos no *Matlab*, 4.9 segundos em *Python* e 5.3 segundos no *R*. Assim se pode verificar que este algoritmo é um pouco mais lento em todas as ferramentas e linguagens usadas, contudo usando um *dataset* de 434874 podemos verificar a escalabilidade do algoritmo que demonstrou um aumento de performance, tendo-se verificado 442 segundos no *Tensorflow*. *MATLAB* demorou 1011 segundos, ao passo que *Python* demorou 913 segundos e *R* demorou 821 segundos.

3.4 Resumo ou Conclusões

É então possível concluir, que o *Tensorflow* apresenta sem dúvida uma ótima perfomance em algoritmos de *clustering* em comparação com as outras ferramentas utilizadas como termo de referência para este trabalho. Podemos também facilmente concluir que o algoritmo *K-Means* foi o algoritmo que mais rapidamente conseguiu alcançar resultados, apesar que durante a pesquisa efetuada antes do início deste trabalho, chegamos à conclusão que o algoritmo *K-Means* é rápido mas por vezes os seus resultados podem não ser os mais fiáveis. O algoritmo *Affinity Propagation* é extremamente mais lento mas produz resultados muito fiáveis. Esta questão é altamente abordada na área da análise de tráfego pois muitas vezes, principalmente em questões de segurança, o *feedback* da análise dos dados da rede de tráfego tem que ser algo praticamente em *real-time*, mantendo um mínimo de fiabilidade nos resultados.

Por fim, o algoritmo *Mean Shift*, apresenta-se como uma espécie de intermédio entre o *K-Means* e o *Affinity Propagation*, pois apesar de com *datasets* relativamente pequenos, é considerado lento mas apresenta uma boa escalabilidade e com *datasets* de grande dimensão apresenta uma performance relativamente boa, mantendo igualmente uma boa fiabilidade de resultados.

É de enfatizar ainda que os resultados obtidos foram recorrendo apenas a uma GPU, como referido anteriormente, e a diferença de resultados poderia ter sido ainda maior se tivesse havido

a oportunidade de tirar partido do potencial de concorrência de execução do *Tensorflow* com mais do que uma GPU.

Capítulo 4

Conclusões e Trabalho Futuro

4.1 Trabalho Realizado

Foi então possível alcançar o desenvolvimento dum método de avaliação do *Tensorflow*, no que toca a algoritmos de *clustering* relativamente a outras ferramentas que satisfazem esse propósito.

Neste sentido após uma extensa procura e pesquisa sobre quais seriam os algoritmos mais adequados para o propósito em questão, foi também necessário especificar exatamente quais as ferramentas que mais se adequariam a pertencer aos testes que iriam ser realizados. É então de frisar que este projeto teve então uma grande componente de pesquisa.

Posteriormente, procedeu-se à implementação dos algoritmos nas diferentes plataformas, o que se revelou algo mais complexo do que o esperado devido à falta de familiaridade com as diferentes ferramentas usadas.

Neste sentido penso que a principal área para a qual esta dissertação contribui é, sem dúvida, para a área de Redes, pois as aplicações da performance do *Tensorflow* na área da Análise de Tráfego são inúmeras, o que poderá encurtar imenso a execução de grande parte dos algoritmos de *clustering*.

Isto por si só poderá trazer grandes benefícios para as empresas na área da segurança, inteligência militar, entre outras aplicações que poderão beneficiar da performance do *Tensorflow*.

4.2 Trabalho Futuro

É no entanto de referir, mais uma vez, que todos os testes foram realizados sem recorrer ao potencial do *Tensorflow* na sua totalidade. Pois este é capaz de dividir as tarefas de um qualquer programa, de grande ou pequena dimensão, pelos dispositivos existentes. Apesar de normalmente um computador possuir vários núcleos de processamento na CPU, o *Tensorflow* tira partido deste facto por definição. Mas na eventualidade de se poder tirar partido de uma máquina com mais do que uma GPU, esta performance, já verificada, poderá ainda aumentar mais pois teríamos o

dobro do poder de processamento. Neste sentido, como trabalho futuro seria, sem dúvida, algo a explorar. A execução de testes e comparações em algoritmos de *clustering* recorrendo a maior poder de processamento e verificar o quanto poderíamos beneficiar com este poder de *hardware*.

Referências

[AAB+15] Martin Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dan Man, Rajat Monga, Sherry Moore, Derek Murray, Jon Shlens, Benoit Steiner, Ilya Sutskever, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Oriol Vinyals, Pete Warden, Martin Wicke, Yuan Yu e Xiaoqiang Zheng. TensorFlow: Large-Scale Machine Learning on Heterogeneous Distributed Systems. *None*, page 19, 2015. URL: http://download.tensorflow.org/paper/whitepaper2015.pdf, arXiv:1603.04467.

[BCA11] Tekin Bicer, David Chiu e Gagan Agrawal. A framework for data-intensive computing with cloud bursting. *Proceedings - IEEE International Conference on Cluster Computing, ICCC*, pages 169–177, 2011. doi:10.1109/CLUSTER.2011.21.

[Cpu] Cpu and gpu. https://www.tensorflow.org/versions/r0.9/how_tos/using_gpu/index.html. Accessed: 2016-06-21.

[Der05] Konstantinos G Derpanis. Mean Shift Clustering. *Computer*, 1(x):1–3, 2005. URL: http://www.cse.yorku.ca/{~}kosta/CompVis{_}Notes/mean{_}shift.pdf, doi:10.1117/12.792998.

[FD10] Francesco Fusco e Luca Deri. High speed network traffic analysis with commodity multi-core systems. *Proceedings of the 10th annual conference on Internet measurement - IMC '10*, page 218, 2010. URL: http://portal.acm.org/citation.cfm?doid=1879141.1879169, doi:10.1145/1879141.1879169.

[Jai10] Anil K. Jain. Data clustering: 50 years beyond K-means. *Pattern Recognition Letters*, 31(8):651–666, 2010. URL: http://dx.doi.org/10.1016/j.patrec.2009.09.011, arXiv:0402594v3, doi:10.1016/j.patrec.2009.09.011.

[JMF+00] A K Jain, M N Murty, P J Flynn, Azriel Rosenfeld, K Bowyer, N Ahuja e A Jain. Data Clustering: A Review. 2000. arXiv:arXiv:1101.1881v2, doi:10.1145/331499.331504.

[MCB13] Michael Mattess, Rodrigo N. Calheiros e Rajkumar Buyya. Scaling MapReduce applications across hybrid clouds to meet soft deadlines. *Proceedings - International Conference on Advanced Information Networking and Applications, AINA*, pages 629–636, 2013. doi:10.1109/AINA.2013.51.

[MGB15] R M, P G e J B. A Scheduler for Cloud Bursting of Map-Intensive Traffic Analysis Jobs. I(1):11–21, 2015.

[ten16] 2016. URL: https://www.tensorflow.org/versions/r0.9/how_tos/using_gpu/index.html.

[WZL+07] Kaijun Wang, Junying Zhang, Dan Li, Xinna Zhang e Tao Guo. Adaptive Affinity Propagation Clustering. *Automatica*, 33(12):1242–1246, 2007. arXiv:0805.1096, doi:10.1360/aas-007-1242.

[XWZZ08] Ding-yin Xia, Fei Wu, Xu-qing Zhang e Yue-ting Zhuang. Local and global approaches of affinity propagation clustering for large scale data. *Journal of Zhejiang University SCIENCE A*, 9(10):1373–1381, 2008. URL: http://link.springer.com/article/10.1631/jzus.A0720058$\delimiter"026E30F$nhttp://link.springer.com/article/10.1631{%}2Fjzus.A0720058?LI=true, arXiv:0910.1650, doi:10.1631/jzus.A0720058.